METHOD FOR USING EXTREMA TO LOAD BALANCE A
LOOP OF PARALLEL PROCESSING ELEMENTS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application is related to U.S. Patent Application Serial No._____ entitled "Method for Load Balancing a Loop of Parallel Processing Elements" filed _____ (DB001068-000, Micron no. 02-1298), U.S. Patent Application Serial No. _____ entitled "Method for Using Filtering to Load Balance a Loop of Parallel Processing Elements" filed _____ (DB001061-000, Micron no. 02-1296), U.S. Patent Application Serial No. _____, entitled "Method for Load Balancing a Line of Parallel Processing Elements" filed _____ (DB001066-000, Micron no. 02-1267), U.S. Patent Application Serial No._____ entitled "Method for Load Balancing an N-Dimensional Array of Parallel Processing Elements" filed _____ (DB001062-000, Micron no. 02-1295) , U.S. Patent Application Serial No._____ entitled "Method for Rounding Values for a Plurality of Parallel Processing Elements" filed _____ (DB001064-000, Micron no. 02-1269), and U.S. Patent Application Serial No._____ entitled "Method of Obtaining Interleave Interval for Two Data Values" filed _____ (DB001065-000, Micron no. 02-1268).

BACKGROUND OF THE INVENTION

[0002] The present invention relates generally to parallel processing and more particularly to balancing the work loads of the processing elements within a parallel processing system.

[0003] Conventional central processing units ("CPU's"), such as those found in most personal computers, execute a single program (or instruction stream) and operate on a single stream of data. For example, the CPU fetches its program and data from a random access memory ("RAM"), manipulates the data in accordance with the program instructions, and writes the results back sequentially. There is a single stream of instructions and a single stream of data (note: a single operation may operate on more than one data item, as in X = Y + Z, however, only a single stream of results is produced). Although the CPU may determine the sequence of instructions executed in the program itself, only one operation can be completed at a time. Because conventional CPUs execute a single program (or instruction stream) and operate on a single stream of data, conventional CPUs may be referred to as a single-instruction, single data CPU or an SISD CPU.

[0004] The speed of conventional CPUs has dramatically increased in recent years. Additionally, the use of cache memories enables conventional CPUs faster access to the desired instruction and data streams. However because conventional CPUs can complete only one operation at a time, conventional CPUs are not suitable for extremely demanding applications having large data sets (such as moving image processing, high quality speech recognition, and analytical modeling applications, among others).

[0005] Improved performance over conventional SISD CPUs may be achieved by building systems which exhibit parallel processing capability. Typically, parallel processing systems use multiple processing units or processing elements to simultaneously perform one or more tasks on one or more data streams. For example in one class of parallel processing system, the results of an operation from a first CPU are passed to a second CPU for additional processing, and from the second CPU to another CPU, and so on. Such a system, commonly known as a "pipeline", is referred to as a multiple-instruction, single-data or MISD system because each CPU receives a different instruction stream while operating on a single data stream. Improved performance may also be obtained by using a system which contains many autonomous processors, each running its own program (even if the program running on the processors is the same code) and producing multiple data streams. Systems in this class are referred to as a multiple-instruction, multiple-data or MIMD system.

[0006] Additionally, improved performance may be obtained using a system which has multiple identical processing units each performing the same operations at once on different data streams. The processing units may be under the control of a single sequencer running a single program. Systems in this class are referred to as a single-instruction, multiple data or SIMD system. When the number of processing units in this type of system is very large (e.g., hundreds or thousands), the system may be referred to as a massively parallel SIMD system.

[0007] Nearly all computer systems now exhibit some aspect of one or more of these types of parallelism. For example, MMX extensions are SIMD; multiple processors (graphics processors, etc) are MIMD; pipelining (especially in graphics accelerators) is MISD. Furthermore, techniques such as out of order execution and multiple execution units have been used to introduce parallelism within conventional CPUs as well.

[0008] Parallel processing is also used in active memory applications. An active memory refers to a memory device having a processing resource distributed throughout the memory structure. The processing resource is most often partitioned into many similar processing elements (PEs) and is typically a highly parallel computer system. By distributing the processing resource throughout the memory system, an active memory is able to exploit the very high data bandwidths available inside a memory system. Another advantage of active memory is that data can be processed "on-chip" without the need to transmit the data across a

system bus to the CPU or other system resource. Thus, the work load of the CPU may be reduced to operating system tasks, such as scheduling processes and allocating system resources.

[0009] A typical active memory includes a number of interconnected PEs which are capable of simultaneously executing instructions sent from a central sequencer or control unit. The PEs may be connected in a variety of different arrangements depending on the design requirements for the active memory. For example, PEs may be arranged in hypercubes, butterfly networks, one-dimensional strings/loops, and two-dimensional meshes, among others.

[0010] In typical active memories, load imbalances often occur such that some PEs are idle (i.e., without assigned tasks) while other PEs have multiple tasks assigned. To maximize the effectiveness of the active memory, it is desirable to balance the work load across all of the PEs. For example in an active memory having a multitude of identical PEs, it is desirable that each PE be assigned the same number of instructions by the central sequencer, thus maximizing the resources of the active memory. Additionally in an active memory having non-identical PEs, it may be desirable to assign more tasks to the PEs with greater processing capabilities. By balancing the load, the amount of time that one or more PEs is idle while waiting for one or more other PEs to complete their assigned tasks is minimized.

[0011] Thus, there exists a need for a method for balancing the load of a parallel processing system such that the resources of the parallel processing system are maximized. More specifically, there exists a need for a method for balancing the load of an active memory such that the resources of the active memory are maximized.

SUMMARY OF THE INVENTION

[0012] One aspect of the present invention relates to a method for balancing the load of a parallel processing system having a plurality of parallel processing elements arranged in a loop, wherein each processing element (PE_r) has a local number of tasks associated therewith, wherein r represents the number for a selected processing element, and wherein each of the processing elements is operable to communicate with a clockwise adjacent processing element and with an anti-clockwise adjacent processing element. The method comprises determining a total number of tasks present within the loop, calculating a local mean number of tasks for each of the plurality of processing elements, calculating a local deviation for each of the plurality of processing elements, determining a running partial deviation sum for each of the plurality of processing elements, determining a clockwise transfer parameter and an anti-clockwise transfer parameter for each of the plurality of processing elements, and redistributing tasks among the plurality of processing elements in response to the clockwise

3

transfer parameter and the anti-clockwise parameter for each of the plurality of processing elements.

[0013] Another aspect of the present invention relates to a method for reassigning tasks associated with a selected processing element within a parallel processing system having a plurality of processing elements connected in a loop, each of the plurality of processing elements having a local number of tasks associated therewith. The method comprises determining the total number of tasks on the loop, computing a local mean value for the selected processing element, computing a local deviation for the selected processing element, the local deviation representative of the difference between the local number of tasks for the selected processing element and the local mean value for the selected processing element, determining a running partial deviation sum for the selected processing element, computing a number of tasks to transfer in a clockwise direction for the selected processing element, computing a number of tasks to transfer in an anti-clockwise direction for the selected processing element, and reassigning the tasks associated with the selected processing element relative to the number of tasks to transfer in a clockwise direction and the number of task to transfer in an anti-clockwise direction.

[0014] The present invention enables tasks to be distributed along a group of serially connected PEs so that each PE typically has X number of tasks or (X+1) number of tasks to perform in the next phase. The present invention may be performed using the hardware and software (i.e., the local processing capability) of each PE within the array. Those advantages and benefits, and others, will become apparent from description of the invention below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] To enable the present invention to be easily understood and readily practiced, the present invention will now be described for purposes of illustration and not limitation, in connection with the following figures wherein:

[0016] FIG. 1 is a block diagram illustrating an active memory according to an embodiment of the present invention.

[0017] FIG. 2 is a block diagram of a processing element for the active memory as illustrated in FIG. 1 according to an embodiment of the present invention.

[0018] FIG. 3 illustrates an array of the processing elements as illustrated in FIG. 2 arranged in a loop according to an embodiment of the present invention.

[0019] FIG. 4 illustrates an operational process for balancing the load within a loop of processing elements according to an embodiment of the present invention.

[0020] FIGS. 5 and 6 illustrate the relationship between transfer rates around a loop of processing elements and a local transfer rate according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0021] As discussed above, parallel processing systems may be placed within one or more classifications (e.g., MISD, MIMD, SIMD, etc.). For simplicity, the present invention is discussed in the context of a SIMD parallel processing system. More specifically, the present invention is discussed in the context of a SIMD active memory. It should be noted that such discussion is for clarity only and is not intended to the limit the scope of the present invention in any way. The present invention may be used for other types and classifications of parallel processing systems.

[0022] FIG. 1 is a block diagram illustrating an active memory 10 according to an embodiment of the present invention. It should be noted that the active memory 10 is only one example of a device on which the methods of the present invention may be practiced and those of ordinary skill in the art will recognize that the block diagram of FIG. 1 is an overview of an active memory device 10 with a number of components known in the art being omitted for purposes of clarity.

[0023] Active memory 10 is intended to be one component in a computer system. Processing within active memory 10 is initiated when the active memory 10 receives commands from a host processor (not shown), such as the computer system's CPU. A complete processing operation (i.e., data movement and processing) in the active memory 10 may consist of a sequence of many commands from the host to the active memory device 10.

[0024] Active memory 10 is comprised of a host memory interface ("HMI") 12, a bus interface 14, a clock generator 16, a task dispatch unit ("TDU") 18, a DRAM control unit ("DCU") 20, a DRAM module 22, a programmable SRAM 24, an array control sequencer 26, and a processing element array 28, among others.

[0025] The HMI 12 provides an input/output channel between the host (such as a CPU, not shown) and the DRAM module 22. In the current embodiment, the HMI 12 receives command (cmd), address (addr), and data signals (among others) from and sends data and ready (rdy) signals (among others) to the host. The HMI 12 approximates the operation of a standard non-active memory so that the host, without modifications, is compatible with the active memory 10.

[0026] The HMI 12 may be similar in its operation to the interface of a synchronous DRAM as is know in the art. Accordingly, the host must first activate a page of data to access data within a DRAM module 22. In the current embodiment, each page may contain 1024 bytes of

5

data and there may be 16,384 pages in all. Once a page has been activated, it can be written and read through the HMI 12. The data in the DRAM module 22 may be updated when the page is deactivated. The HMI 12 also sends control signals (among others) to the DCU 20 and to the processing element array 28 via the task dispatch unit 18.

[0027] The HMI 12 may operate at a frequency different than that of the frequency of the master clock. For example, a 2x internal clock signal from clock generator 16 may be used. Unlike a traditional DRAM, the access time for the HMI 12 uses a variable number of cycles to complete an internal operation, such as an activate or deactivate. Thus the ready signal (rdy) is provided to allow the host to detect when a specific command has been completed.

[0028] The bus interface 14 provides and input/output channel between the host and the TDU 18. For example, the bus interface 14 receives column select (cs), write command (w), read command (r), address (addr), and data signals (among others) from and places interrupt (intr), flag, and data signals (among others) onto the system bus (not shown). The bus interface 14 also receives signals from and sends signals to TDU 18.

[0029] The clock generator 16 is operable to receive an external master clock signal (x1) and operable to provide the master clock signal (x1) and one or more internal clock signals (x2, x4, x8) to the components of the active memory. It should be apparent to one skilled in the art that other internal clock signals may be produced by the clock generator 16.

[0030] The TDU 18 communicates with the bus interface 14, the HMI 12, the programmable SRAM 24, the array control sequencer 26, and the DCU 20. In the current embodiment, the TDU 18 functions as an interface to allow the host to issue a sequence of commands to the array control sequencer 26 and the DCU 20. Task commands from the host may be buffered in the TDU's FIFO buffers to allow a burst command to be issued. Commands may contain information on how the tasks in the array control sequencer 26 and the DCU 20 should be synchronized with one another, among others.

[0031] The DCU 20 arbitrates between the TDU 18 and the HMI 12 and sends commands to the DRAM modules 22 and the processing element array 28. The DCU 20 also schedules refreshes within the DRAM modules 22. In one embodiment, the DRAM modules 22 of the active memory 10 may be comprised of sixteen 64k x128 eDRAM (or embedded DRAM) cores. Each eDRAM core may be connected to an array of sixteen PEs, thus providing 256 (16 x 16) PEs in all.

[0032] The programmable SRAM 24 functions as a program memory by storing commands issued by the TDU 18. For example, the TDU 18 may transmit a "write program memory address" command which sets up a start address for a write operation and a "write program memory data" command which writes a memory location and increments the program

memory write address, among others. The programmable SRAM 24, in the current embodiment, has both an address register and a data output register.

[0033] The array control sequencer 26 may be comprised of a simple 16 bit minimal instruction set computer (16-MISC). The array control sequencer 26 communicates with the TDU 18, the programmable SRAM 24, and the DCU 20, and is operable to generate register file addresses for the processing element array 28 and operable to sequence the array commands, among others.

[0034] The processing element array 28 is comprised of a multitude of processing elements ("PEs") 30 (see FIG. 2) connected in a variety of different arrangements depending on the design requirements for the processing system. For example, processing units may be arranged in hypercubes, butterfly networks, one-dimensional strings/loops, and two-dimensional meshes, among others. In the current embodiment, the processing elements 30 are arranged in a loop (for example, see FIG. 3). The processing element array 28 communicates with the DRAM module 22 and executes commands received from the programmable SRAM 24, the array control sequencer 26, the DCU 20, and the HMI 12. Each PE in the processing element array 28 includes dedicated H-registers for communication with the HMI 12. Control of the H-registers is shared by the HMI 12 and the DCU 20.

[0035] Referring now to FIG. 2, a block diagram of a PE 30 according to one embodiment of the present invention is illustrated. PE 30 includes an arithmetic logic unit ("ALU") 32, Q-registers 34, M-registers 36, a shift control and condition register 38 (also called "condition logic" 38), a result register pipeline 40, and register file 42. The PE 30 may also contain other components such as multiplexers 48 and logic gates (not shown), among others.

[0036] In the current embodiment, the Q-registers 34 are operable to merge data into a floating point format and the M-Registers 36 are operable to de-merge data from a floating point format into a single magnitude plus an exponent format. The ALU 32 is a multiplier-adder operable (among others) to receive information from the Q-registers 34 and M-registers 36, execute tasks assigned by the TDU 18 (see FIG. 1), and transmit results to the shift control and condition logic 38 and to the result register pipeline 40. The result register pipeline 40 is operable to communicate with the register file 42, which holds data for transfer into or out of the DRAM modules 22 via a DRAM interface 44. Data is transferred between the PE and the DRAM module 22 via a pair a registers, one register being responsive to the DCU 20 and the other register being responsive to the PE 30. The DRAM interface receives command information from the DCU 20. The DRAM interface 44 also permits the PE 30 to communicate with the host through the host memory access port 46.

[0037] In the current embodiment, the H-registers 42 are comprised of synchronous SRAM and each processing element within the processing element array 28 contains eight H-registers

7

42 so that two pages can be stored from different DRAM locations, thus allowing the interleaving of short i/o bursts to be more efficient. Result register pipeline 40 is also connected to one or more neighborhood connection registers ("X-register") (not shown). The X-register links one PE 30 to its neighboring PE's 30 in the processing element array 28.

[0038] The reader desiring more information about the hardware shown in FIGs. 1 and 2 is directed to UK Patent application (serial no. not yet assigned) entitled "Control of Processing Elements in Parallel Processors" filed 17 September 2002, (Micron no. 02-1604) which is hereby incorporated by reference. Details about the PEs may also be found in UK Patent Application No. 021562.2 entitled "Host Memory Interface for a Parallel Processor" filed 17 September 2002, (Micron no. 02-0703) which is hereby incorporated by reference.

[0039] FIG. 3 is a simplified diagram showing the interconnections of an array of the processing elements 30 (as illustrated in FIG. 2) arranged in a loop 50 according to an embodiment of the present invention. In the current embodiment, loop 50 is comprised of eight (8) PEs 30 (i.e., PE_0, PE_1, ... PE_7) which are interconnected via their associated X-register links. It should be noted that the number of PEs 30 included in loop 50 may be altered while remaining within the scope of the present invention. As illustrated in FIG. 3, each PE is operable to communicate with its clockwise and anti-clockwise neighbor. For example, PE_1 is operable to communicate with its clockwise neighbor, PE_2, and with its anti-clockwise neighbor, PE_0. In the current embodiment, every PE 30 on the loop 50 receives instructions from a single TDU 18 as discussed in conjunction with FIG. 1. Furthermore, each PE has a local number of tasks (v_r) associated therewith. For example, PE_0 has three (3) tasks associated therewith (i.e., $v_0 = 3$), PE_1 has six (6) tasks associated therewith (i.e., $v_1 = 6$), PE_2 has two (2) tasks associated therewith (i.e., $v_2 = 2$), etc.

[0040] FIG. 4 illustrates an operational process 60 for balancing the work load between the PEs 30 on loop 50 according to an embodiment of the present invention. Operational process 60 begins by determining the total number of tasks (V) present on the loop in operation 61. As discussed above in conjunction with FIG. 3, each PE_r (where $r = 0$ to 7, e.g., PE_0, PE_1, ... PE_7) in the loop has a local number of tasks (v_r) associated therewith. In the current embodiment, each PE_r passes its own value v_r onto its anti-clockwise neighbor and simultaneously receives a value v_{r+1} from its clockwise neighbor. Each PE_r keeps a running partial sum (i.e., adds each value v_{r+1} received to its own value v_r). This process continues until each value v_r has moved anti-clockwise around the loop and visited each PE_r, in this case seven transfers are needed. At the end of the rotation process, the sum represents the total number of tasks (V) on the loop.

[0041] The sum (V) can be expressed by the equation $V = \sum_{i=0}^{i=N-1} v_i$, where N represents the

number of PEs 30 in the loop 50 (here $N = 8$), and v_i represents the local number of tasks

associated with an ith processing element in the loop. For example, for $i = 3$, the number of

tasks associated with PE$_3$ (i.e., v_3) is added to the sum V. It should be noted that after a

rotation is completed, each PE$_r$ will have calculated the same value for (V). It should also be

noted in the current discussion, "local" refers to the values or functions associated with a

single PE within the loop, whereas "global" refers to the values or function associated with

the entire loop of PEs.

[0042] After the total number of tasks (V) present on the loop is determined in operation 61,

the local mean number (M_r) of tasks for each PE$_r$ is computed in operation 62. In the current

embodiment, operation 62 employs a rounding function to ensure that no tasks are lost or

"gained" during the rounding process (i.e., to ensure that $V = \sum_{i=0}^{i=N-1} M_i$).

[0043] For example assume that 13 tasks (i.e., $V = 13$) are to be shared by the eight PEs (i.e.,

PE$_0$ through PE$_7$). Without the rounding function, the local mean for each PE would be PE$_r$ =

1.625 before rounding (i.e., $13 \div 8 = 1.625$). If the fraction thirteen-eighths is set to round

down for each PE (i.e., $13 \div 8 = 1$), then the sum of the means for all of the individual PEs

(i.e., PE$_0$ through PE$_7$) is equal to eight (8) and five tasks ($13 - 8 = 5$) are lost. In contrast, if

the fraction thirteen-eighths is set to round up for each PE (i.e., $13 \div 8 = 2$), then the sum of

the means for all of the individual PEs (i.e., PE$_0$ through PE$_7$) is equal to sixteen (16) and

three extra tasks ($16 - 13 = 3$) are gained.

[0044] In the current embodiment, the rounding function $M_r = Trunc((V + E_r)/N)$ is

employed, which prevents tasks from being lost or gained (where M_r represents the local

mean for PE$_r$, N represents the total number of PEs 30 in the loop 50, and E_r represents a

number in the range of 0 to ($N-1$)). The rounding function used by the current embodiment

is discussed in more detail in U.S. Patent Application Serial No._____ entitled "Rounding

Algorithm for a Plurality of Parallel Processing Elements" filed _____ (DB001064-000,

Micron no. 02-1269) and incorporated in its entirety by reference herein.

[0045] In the current embodiment, each PE is assigned a different E_r value for controlling the

rounding. The simplest form for the function E is the case in which $E_r = P_r$, where P_r

represents the PEs position in the loop. For example, for PE$_0$, E$_0$ = 0; for PE$_1$, E$_1$ = 1; for PE$_2$,

E$_2$ = 2; etc. By assigning each PE in the loop a different E_r value, the rounding function can

be controlled such that some of the local means are rounded up and some of the local means

are rounded down, thus insuring that $V = \sum\limits_{i=0}^{i=N-1} M_i$. It should be noted that in the current

embodiment, the local mean for each PE 30 in the loop is computed in parallel with the local means of the other PEs in the loop.

[0046] Table 1 illustrates the local mean calculation for the loop 50 as illustrated in FIG. 3 in which the total number of tasks on the loop is equal to forty-three (43). Referring to Table 1, it is apparent that the rounding function controls the rounding such that M_0 through M_4 are all rounded to five (5), whereas M_5 through M_7 are all rounded to six (6) . The sum of the values of M_0 through M_7 is equal to forty-three (43), which equals the total number of tasks (V) on the loop. Thus, tasks are neither lost nor gained due to rounding.

PE_r	v_r	E_r	$(V+E_r)/N$	$M_r = Trunc((V+E_r)/N)$	D_r
PE_0	3	0	5.375	5	-2
PE_1	6	1	5.5	5	1
PE_2	2	2	5.625	5	-3
PE_3	7	3	5.75	5	2
PE_4	8	4	5.875	5	3
PE_5	5	5	6.0	6	-1
PE_6	5	6	6.125	6	-1
PE_7	7	7	6.25	6	1

Table #1 – Local Mean Calculation for the Loop 50 ($V = 43$, $N = 8$).

[0047] After the local means are computed in operation 62, the local deviation D_r is calculated for each PE_r in operation 63. In the current embodiment, the local deviation is for PE_r is simply the difference between the local number of tasks and the local mean (i.e., $D_r = v_r - M_r$). The local deviations D_r for PE_0 through PE_7 are illustrated in Table #1.

[0048] After the local deviations are computed in operation 63, a running partial deviation sum (S_j) is determined for each PE_r in operation 64. The partial deviation sum (S_j) is formed in a similar manner as that used to form the partial value sum (V) in operation 61. In operation 64, however, the local deviations (D_r) are summed instead of the local number of tasks (v_r). If "j" deviations have been moved around the loop, then the partial deviation sum

(S_j) can be represented by the equation: $S_j = \sum\limits_{i=0}^{i=j} D_i$.

10

[0049] It should be noted that the sum of the deviations of all the PEs on the loop 50 is zero (i.e., $S_{N-1} = 0$). In other words, if $j = (N-1)$, then $S_j = 0$. Referring to Table #1 for example, the partial deviation sum relative to PE_0 where $j = (N-1)$ (i.e., S_{N-1}) is $D_0 + D_1 + D_2 + D_3 + D_4 + D_5 + D_6 + D_7 = (-2) + 1 + (-3) + 2 + 3 + (-1) + (-1) + 1 = 0$. In general however, the local partial deviation sum (S_j) will be different for each PE_r if $j \neq (N-1)$. Referring to Table #1 for example, the partial deviation sum relative to PE_1 where $j = 3$ (i.e., S_3) is $D_1 + D_2 + D_3 + D_4 = 1 + (-3) + 2 + 3 = 3$. Whereas, the partial deviation sum relative to PE_4 where $j = 3$ (i.e., S_3) is $D_4 + D_5 + D_6 + D_7 = 3 + (-1) + (-1) + 1 = 2$.

[0050] After a running partial deviation sum (S_j) is determined for each PE_r in operation 64, an anti-clockwise transfer parameter (T_a) and a clockwise transfer parameter (T_c) are determined for each PE_r in operation 65. In the current embodiment, each PE_r keeps a record of the highest (H_r) and lowest (L_r) sums (S_j) encountered in operation 64. The highest (H_r) and lowest (L_r) sums (S_j) may be referred to as extrema and are generally different for each PE_r even after all deviations have been summed. Each PE_r determines its own transfer parameters from the local (H_r) and local (L_r) values. For example in the current embodiment, the anti-clockwise transfer parameter (T_a) and the clockwise transfer parameter (T_c) are determined using the equations $T_a = (H_r + L_r) \div 2$ and $T_c = D - T_a$, respectively.

[0051] Referring to FIGS. 5 and 6, PE_r has an initial deviation of P_r and a_r is the number of tasks received clockwise from PE_{r-1} (situated immediately clockwise) and a_{r+1} is the number of tasks sent out by PE_r in a clockwise direction. After the transfer of these tasks, all PEs will have zero deviation. Thus, $P_r + a_r - a_{r+1} = 0$ for all $0 \leq r < n$. Furthermore, $a_1 = a_0 + P_0$; $a_2 = a_1 + P_1$; $a_3 = a_2 + P_2$; ... etc. Thus, the general equation $a_r = a_0 + \sum_{i=0}^{i=r-1} P_i$ may be derived.

[0052] As the values P_i are moved around the loop each PE_r can evaluate its local version of general equation above. Each PE_r will have $(n-1)$ equations expressing a_r in terms of a_0 and the partial sum of deviations. For example, $a_1 = a_0 + S_1$; $a_2 = a_0 + S_2$; $a_3 = a_0 + P_3$; ... $a_{n-1} = a_0 + S_{n-1}$. For any given PE_r, the transfer rates for every step around the loop can be related to the local transfer value a_0. Referring to FIG. 6, the $(n-1)$ equations for a given PE_r may be plotted as a series of parallel lines. In order to minimize the transfer rates the value of a_0 is chosen to make the maximum positive excursion the same as the maximum negative excursion. This value is illustrated in FIG 6 as a_e. Accordingly, the local transfer values are represented by the equations $a_0 = a_e$ and $a_1 = a_e + P_0$.

[0053] To find the actual value for a_e, it is apparent from FIG. 6 that the two extrema H_r and L_r (represented in FIG. 6 as S_1 and S_3, respectively) need to be determined. H_r and L_r may be referred to as MAX and MIN, respectively. Thus referring to FIG. 6,

11

$$a_e = \frac{-(MAX + MIN)}{2}$$. It should be noted that MAX and MIN are the maximum and

minimum values of $\sum\limits_{i=0}^{i=r-1} P_i$ as r moves from $r = 0$ to $r = n - 1$. As each PE$_r$ starts with a

different value of P_0, the values for MAX, MIN, and a_e will also be different for each PE$_r$.
The values MAX and MIN can be updated "on-the-fly" as the deviations are moved round the

loop and the sums $\sum\limits_{i=0}^{i=r-1} P_i$ are accumulated. Thus, the values obtained for H$_r$ and L$_r$ may be

used to calculate the clockwise transfer parameter (T_c) and the anti-clockwise transfer

parameter (T_a), where $T_a = -a_0$ and $T_c = a_1$.

[0054] It should be noted that the values obtained for T_c and T_a may need to be rounded in
such a manner that $R(T_c) + R(T_a) = P_r$. In the current embodiment, tasks are transmitted in
only one direction at a time around the loop (i.e., either in the clockwise or anti-clockwise
direction). A direction is selected for the 'first' transmission around the loop and the values
for T_c and T_a are rounded up in this direction. It should be noted that by ensuring 'excess
traffic' is sent in the 'first' direction, the chance of the process finishing one step earlier is
increased. In the current embodiment, tasks are transmitted in the anti-clockwise first, such
that $R(T_a) = Ceil\ (T_a)$, where the 'Ceil' function returns the closest integer greater than or
equal to the supplied input. To ensure that extra tasks are not created or lost by the rounding
of $R(T_a)$, $R(T_c)$ is set equal $P_r - R(T_a)$.

[0055] It should be noted that other rounding mechanisms may be used while remaining
within the scope of the present invention. For example, T$_c$ may be rounded up on odd
numbered PE's and T$_a$ rounded up on even numbered PE's such that pairs of odd and even
PE's exchange their 'excess traffic'.

[0056] In the case where the loop 50 is comprised of an odd number of PEs 30, an extra
"phantom" PE may be used. The phantom PE is assigned a deviation of zero (0) and is
located diametrically opposite from the perspective of the selected PE. For example, assume
that PE$_7$ is removed from loop 50 such that only seven PEs remain (i.e., PE$_0$ to PE$_6$). To
calculate the local deviation of PE$_0$, the phantom PE would be placed between PE$_3$ and PE$_4$;
for PE$_1$, between PE$_4$ and PE$_5$; for PE$_2$, between PE$_5$ and PE$_6$, etc. Thus, the phantom PE is
located in the loop 50 such that the number of PEs between the selected PE and the phantom
PE in the clockwise direction is equal to the number of PEs between the selected PE and the
phantom PE in the anti-clockwise direction.

[0057] After the clockwise transfer parameter (T_c) and the anti-clockwise transfer parameter
(T_a) are determined for each PE$_r$ in operation 65, the tasks are redistributed among the PEs
within the loop in response to the clockwise transfer and anti-clockwise transfer parameters

12

(T_c and T_a, respectively) in operation 66. In the current embodiment, a positive T_c parameter represents the number of values that are to be transmitted clockwise out of the local PE. A negative T_c parameter represents the number of values that are to be transmitted from the clockwise PE into the local PE. Similarly, a positive T_a parameter represents the number of values that are to be transmitted anti-clockwise out of the local PE. A negative T_a parameter represents the number of values that are to be transmitted from the anti-clockwise PE into the local PE.

[0058] If the local deviation (D) is negative, one or more of the received values will be "absorbed" by the local PE to make up the local deficit. The other will be transmitted, either from the clockwise PE to the anti-clockwise PE, or from the anti-clockwise PE to the clockwise PE. On occasion, some PEs may start off with no values at all, these PEs may have to "mark time" until they receive a value. It should be noted that after each successful transmission or receipt, the local parameters T_c and T_a need to be updated. The redistribution stage only terminates when $T_c = T_a = 0$ for all PEs.

[0059] It should be recognized that the above-described embodiments of the invention are intended to be illustrative only. Numerous alternative embodiments may be devised by those skilled in the art without departing from the scope of the following claims.